CLAIMS:

1. A method for analyzing software code comprising the steps of:

a) automatically generating program graphs representing said code utilizing static analysis techniques;

b) automatically applying a set of rules to said program flow analysis graphs;

c) automatically identifying potential software problems from rules set analysis results; and,

d) reporting said software problems where one or more of best practices violations and coding errors may occur. control and data flow analysis

2. The method according to Claim 1, wherein said rules set represents one or more selected from the group comprising: use of best practices and common coding errors, or combinations thereof.

3. The method according to Claim 1, wherein said reporting d) includes presenting the results in the context of corresponding source code or object code.

4. The method according to Claim 1, wherein step b) includes performing rule searches applied to said program graphs.

5. The method according to Claim 1, wherein said software code subject to said static analysis techniques comprises one or more selected from the group comprising: object code, source code, a compiler intermediate representation, of said software code, and other program representations, or combinations thereof.

6. The method according to Claim 3, wherein a program graph includes a control analysis graph, said static analysis technique automatically generating said control analysis graphs from said software code.

7. The method according to Claim 3, wherein a program graph includes a data flow analysis graph, said static analysis technique automatically generating said data flow analysis graph from said software code.

8. The method according to Claim 3, wherein a program graph includes an intraprocedural control graph, said static analysis technique automatically generating said intraprocedural control graphs from said software code.

9. The method according to Claim 3, wherein a program graph includes an interprocedural control graphs, said static analysis technique includes automatically generating said interprocedural control graphs from said software code.

10. The method according to Claim 5 wherein said static code analysis further includes automatically identifying classes, fields, methods and class attributes, said set of rules being further applied to said classes and class attributes.

11. The method according to Claim 5 wherein said static code analysis further includes automatically identifying attributes of classes, methods, fields, and aspects of a program's body.

12. The method according to Claim 5, wherein said step b) further includes the step of: receiving said program graphs and class attributes information and performing a graph rewriting technique.

13. The method according to Claim 12, wherein a result of applying graph rewriting includes generating a run-time characteristics model for said program.

14. The method according to Claim 12, wherein said step b) further includes the step of receiving said program graphs and attributes information, and performing a reachability analysis.

15. The method according to Claim 14, wherein said reachability analysis is performed with or without constraints.

16. The method according to Claim 14, further comprising the step of employing a rule search engine to automatically apply a set of rules to said rewrite graph results, reachability analysis results and attributes to identify one or more selected from the group of: possible performance errors or problems concerning correctness, security, privacy and maintainability of said software code.

17. The method according to Claim 14, wherein said rewrite graph technique includes traversing a program graph to locate nodes containing attributes of interest and to locate edges to add or remove from said program graph.

18. The method according to Claim 17, wherein said reachability analysis includes traversing the program graphs and adding or removing edges to extend or reduce reachability, respectively.

19. The method according to Claim 18, wherein a rule is applied to determine whether a node representing a particular method is reachable by traversing said graph from a particular head node, said head node being user selectable.

20. A static analysis framework for analyzing software code, said framework comprising:

means for automatically generating program graphs;

rule search engine for automatically applying a set of rules to said program graphs;

means for automatically identifying potential software problems from rules set analysis results; and, means for reporting said problems to enable correction of instances where one or more of best practices violations and common coding errors may occur.

21. The static analysis framework as claimed in Claim 20, wherein said rules set represents one or more selected from the group comprising: use of best practices and common coding errors, or combinations thereof.

22. The static analysis framework as claimed in Claim 20, wherein said software code comprises scalable componentized applications according to a software development platform.

23. The static analysis framework as claimed in Claim 18, wherein said program graphs include one or more selected from the group comprising: a control analysis graph, a data flow analysis graph, an intraprocedural control flow graph and an interprocedural control flow graph, said static analysis technique automatically generating a respective one of said control analysis graph, data flow analysis graph, intraprocedural control flow graph and interprocedural control flow graph from said software code.

24. The static analysis framework as claimed in Claim 23, further including means for automatically identifying classes, fields, methods and class attributes, said set of rules being further applied to said classes and class attributes.

25. The static analysis framework as claimed in Claim 23, wherein said static code analysis further includes automatically identifying attributes of classes, methods, fields, and aspects of a program's body.

26. The static analysis framework as claimed in Claim 20, wherein said means for automatically generating program graphs includes means for performing graph rewriting.

27. The static analysis framework as claimed in Claim 26, wherein results of said graph rewriting include a run-time characteristics model for said program.

28. The static analysis framework as claimed in Claim 26, wherein said means for automatically generating program graphs includes: means for performing a reachability analysis, said reachability analysis being performed with or without constraints.

29. The static analysis framework as claimed in Claim 28, wherein said rule search engine automatically applies a set of rules to said rewrite graph results, reachability analysis results and attributes to identify one or more of: possible performance errors or problems concerning correctness, security and privacy of said software code.

30. A computer program device readable by a machine, tangibly embodying a program of instructions executable by a machine to perform method steps for analyzing software code, said method steps comprising:

a) automatically generating program graphs representing said code utilizing static analysis techniques;

b) automatically applying a set of rules to said program graphs;

c) automatically identifying potential software problems from rules set analysis results; and,

d) reporting said software problems to enable correction of instances where one or more of best practices violations and common coding errors may occur.